# Bacsy Documentation

*Release 0.1*

**Ives van der Flaas**

March 03, 2012

# CONTENTS

CHAPTER

ONE

# GETTING STARTED

CHAPTER

TWO

# WHAT IS BACSY?

Bacsy is a *backup system* (hence the name).

Essential points:

- **Distributed** A device using Bacsy to backup its data will send the backups to other devices also running Bacsy.
- **Easy to add a new device** In a default configuration, if you add a new device running Bacsy to your LAN, backups will automatically be stored on that device.
- **Redundant** Backups will usually be stored on multiple devices.
- **Fast to setup ...** A basic configuration should be quick to achieve
- **... yet extremely adjustable** A plethora of configuration options are available.

CHAPTER

THREE

# HOW DOES BACSY WORK?

**Bacsy is made up out of two components:**

- **A Client** The client generates backups and sends them to one or more servers.
- **A Server** A server accepts and stores backups for one or more clients.

**Most systems will have both a client and a server running.** Systems that have no need to generate backups of their own (e.g. dedicated backup servers) will have only a server running.

## 3.1 Client

A client has a number of *sources*. Each source specifies what files should be backed up, how often they should be backed up, how important they are, etc.

When the client has a backup it needs stored, it will ask qualifying servers if they can store this backup. If the servers respond they can, the backup will be sent to them.

## 3.2 Server

A Bacsy server stores backups. Each server has one or more *stores*. A *store* is a location where backups can be stored, usually some directory on some drive. Backups will often be stored to multiple stores on the same server, for redundancy purposes.

CHAPTER

FOUR

# CONTENTS

## 4.1 Configuration Files

### 4.1.1 Introduction

**Note:** Feel free to read *Example Configurations* before you read this document. You probably won't need 95% of what's written here.

#### Basic

The primary way of configuring Bacsy is by writing a set of human readable files in a format called *Cascading Configuration File*, or *CCF*. A Cascading Configuration File is a normal INI file, with some added semantical meaning.

A Cascading Configuration File will contain a number of sections, each of which start with

```
[ sectionName ]
```

where sectionName is some arbitrary string. A section ends when another section starts or the file ends. Each of these sections will contain a number of assignments that look like this:

```
key = some value
```

For example:

```
[ global ]
Priority = 6
HostIdentification = IX_XUBUNTU
```

This fragment defines a section `global`, which contains two keys `Priority` and `HostIdentification`. The key `Priority` in section `global` will then correspond to a value `6`, and the key `HostIdentification` will correspond to a value `IX_XUBUNTU`.

Keys, section names and values are all case sensitive. Spaces may occur in values.

#### Array Keys

If a key ends in `[]`, this key can be defined multiple times in the same section. When more than one value is assigned to this key, the key will correspond to a list of all values assigned to that key in that section. For example:

```
Include[] = /home/ives
Include[] = /home/naomi
```

is used to include both `/home/ives` **and** `/home/naomi`.

**Note:** Keys by the names of `Include[]` and `Include` are unrelated. Only defining `Include[]` once is **not** equivalent with defining `Include`.

### Comments

All lines that start with a `#` or a `;` character will be ignored.

### The Cascading Part

Each CCF may have a section called `global`. When such a section exists, all key-value pairs defined in this section are inherited by every section that is defined in that file. When that same key is defined in a section other than the global section, it overrides the value that was given to it in section `global`.

For example, if you would like to set the default priority for all sources to `3`, you'd write something like:

```
[ global ]
Priority = 3

[ firstSource ]
Include[] = /home/ives
Exclude[] = *.bak


[ anotherSource ]
Priority = 4
```

In this example, `firstSource` will have priority of 3, and `anotherSource` will have a priority of 4 (because the inherited value of `3` was overwritten.

## 4.1.2 Configuration Files

The following configuration files exist in the directory `.bacsy`.

**sources.configuration** This file contains all information relating to the data sources for which backups should be generated.

**stores.configuration** This file contains all information relating to the data stores present on the current system.

### sources.configuration

Every section name, except for `global` (which is optional), defines a new data source. Source names obviously have to be unique. Each source may, but is not required to, contain the following keys:

**Include[]** **Default Value:** (not defined)

**Semantics:** Each time this key is defined, the associated value is a *single directory or file*. This directory (and all its descendants) or file will be backed up when the source is run, unless it matches one or more exclude rules. If the include is a directory, that directory does not match any exclude, but one of its descendants does, only that descendant will be excluded.

**Example 1:**

```
Include[] = /home/ives
Include[] = /home/naomi/
Include[] = /home/john/.bashrc
```

This example will include two directories, `/home/ives` and `/home/naomi/`, and also a single file `/home/john/.bashrc`.

**Exclude[]** **Default Value:** (not defined)

**Semantics:** (This stuff is a lot easier than it might seem, feel free to look at the examples at the bottom first). A file or directory is excluded by this rule, when it matches all of its non-negated subrules and does not match any of its negated subrules.

A subrule can be a simple path (pointing to a file or directory), it can be a path containing wildcards, and it can be a size exclusion subrule. Subrules can be unquoted, in which case only space and backslash (\) are escaped, both of them by prefixing with a backslash (\). They can also be quoted in double quotes ("), in which case both \ and " have to be escaped with a \.

Subrules are separated by the symbol `&` and may be prefixed with a `!` to negate the subrule.

**Example 1:**

```
Exclude[] = "/home/ives/some directory"
Exclude[] = /home/ives/another\ directory
Exclude[] = /home/ives/.vimrc
Exclude[] = /home/ives/*.bak
Exclude[] = *~
```

The first exclude rule will exclude the directory `/home/ives/some directory`. The second exclude rule will exclude the directory `/home/ives/another directory`. The third one will exclude the file `/home/ives/.vimrc`. The fourth one will exclude all files and directories ending in `.bak` in the directory `/home/ives/`'. The fifth and final one will exclude all files ending in ~ (e.g. `bla.txt~`, `data~`).

**Example 2:**

```
Exclude[] = >50MB
Exclude[] = <100B
```

The first rule will exclude all files that are larger than 50 megabytes, the second rule will exclude all files smaller than 100 bytes.

**Example 3:**

```
Exclude[] = *.avi & >500MB
Exclude[] = /home/ives/*.vob & <1GB
Exclude[] = /home/ives/movies & !/home/ives/movies/reallyGoodMovies
```

The first exclusion rule excludes all avi files that are larger than 500MB. The second one excludes all vob files smaller than one gigabyte. The third one excludes all files in `/home/ives/movies`, except for those in `/home/ives/movies/reallyGoodMovies`.

**ExecuteAt** **Default Value:** ?

**Semantics:** Determines when a source will be executed. There can be multiple triggers, separated by the keyword `and`. Some examples:

**Example 1:**

```
ExecuteAt = every week on Wednesday at 20h00
```

Execute the source every week on Wednesday at 8 p.m.

**Example 2:**

```
ExecuteAt = every 3 hours
```

Execute the source every three hours, the first time being immediately after Bacsy starts.

**Example 3:**

```
ExecuteAt = on start and every 1 minute and every day at 07:00h
```

Execute the source every minute, when Bacsy starts, and also every day at 7 a.m.

**Priority** **Default Value:** ?

**Domain:** [0-10]

**Semantics:** A source's priority determines how important this source is. Higher priorities correspond to **lower** priority numbers. E.g. a source with Priority 5 is **less** important than a source with Priority 2. If you find this confusing, I don't blame you, but know that this was done with good reason; priorities in most systems, including operating systems work this way, so it would have been confusing either way.

A run will only be accepted by a store when the source's Priority is higher (thus the number is lower) than the store's *MinPriorityForStoring*.

Use this key and the associated store key MinPriorityForStoring to make sure really important sources take precedence over less important sources when there isn't a lot of storage left.

**HostIdentification** **Default Value:** Your computer's hostname.

**Semantics:** HostIdentification will be used by stores to categorize backups. When running several Bacsy clients on the same computer, set this key in the configuration to make sure things don't get messed up. This can happen easily when running Bacsy in both a Virtual PC and the host PC.

**MinBackups** **Default Value:** 1

**Domain:** [0 - 2 147 483 648[

**Semantics:** The minimum number of stores Bacsy will try to copy the contents of this source to. When this number is not reached, an error will be produced.

**MaxBackups** **Default Value:** 1

**Domain:** [0 - 2 147 483 648[

**Semantics:** The maximum number of stores Bacsy will try to copy the contents of this store to. Bacsy will never store more backups than this. Use this when you have a very large number of stores in the LAN.

**PreferredOrder** **Default Value:** this, other

**Semantics:** The order in which Bacsy will store to stores. `this` refers to this computer and `other` refers to all other computers. `this, other` will cause Bacsy to first try local stores, before resorting to stores connected to other PCs, resulting in speedier backups. `other, this` will make Bacsy connect to other PCs first, resulting in possibly better protected backups (against theft, fire, ...).

**Distribution** **Default Value:** ?

**Semantics:** Determines the distribution (spread) of where backups will be stored. Ceteris paribus, when storing 10 backups on 10 hosts, each of which has 10 stores, setting this option to `spread` will store 1 backup on each host. Setting it to `focus` will store all 10 backups on one host's stores.

**DryPrintRun** **Default Value:** False

**Domain:** { True, False }

**Semantics:** When DryPrintRun is True, instead of sending files to hosts and their stores, the filenames of all files that *would* be sent are printed to standard output.

This option is most often used to fine tune your inclusion/exclusion rules.

**Enabled Default Value:** True

**Domain:** { True, False }

**Semantics:** If Enabled is False, the target won't actually be executed. Use this to (temporarily?) disable targets.

### stores.configuration

Every section name, except for `global` (which is optional), defines a new data store. Store names obviously have to be unique. The following keys can be defined:

**Location Default Value:** (none)

**Semantics:** The base directory in which all backups and associated data will be stored.

**Example 1:**

```
Location = /media/external_drive/backups
```

Will store all backups this store accepts in `/media/external_drive/backups`.

**MinPriorityForStoring Default Value:** ?

**Domain:** [0,10]

**Semantics:** Determines how high a source's priority must be before it is accepted by this store. Only runs from sources with a *Priority* higher than or equal to (and thus a number that is smaller than or equal to) this store's MinPriorityForStoring will be stored.

**MaxRunsBetweenFullBackups Default Value:** 2

**Domain:** [0 - 2 147 483 648[

**Semantics:** The maximum number of non-full backup runs Bacsy will execute before forcing a full run.

**StoreTime Default Value:** 1 year

**Domain:** <number> <unit> where number must be an integer and <unit> can be either day, days, month, months, year, years.

**Semantics:** The Server is free to erase backup runs that are older than this. Note that a month is defined as 31 days and a year as 366 days (maximal values were chosen to ensure users aren't surprised their data is gone).

**AlwaysPresent Default Value:** True

**Domain:** { True, False }

**Semantics:** When AlwaysPresent is True, Bacsy will assume this store is permanently connected. If the directory given in `Location` does not exist, it will be created. When it cannot be created, an error will be produced.

If AlwaysPresent is False and the directory in `Location` cannot be found, this directory will not be created and the store will be silently ignored.

**Enabled Default Value:** True

**Domain:** { True, False }

**Semantics:** If a store is disabled, nothing will ever be stored to it. Use this switch to keep the configuration for a store, without actually using it.

## 4.2 Building

### 4.2.1 Compiling the source

#### Dependencies

- Poco
- librsync
- cmake
- make
- A C++ compiler

#### Compiling

First get the source by executing

```
git clone git://github.com/Ivesvdf/Bacsy.git
```

or an equivalent git command.

Navigate into the newly created directory with

```
cd Bacsy
```

We'll be doing an out-of-source build, so

```
mkdir build
cd build
```

Then

```
cmake ..
make
```

When all your dependencies are installed, this should go smoothly. Currently, no installer exists.

#### Building Packages

All of these commands are executed from the `build/` directory, after compiling.

##### Building shell and tar.gz packages

Execute

```
cpack
```

##### Building Debian packages

**Warning:** When building Debian packages, it is strongly recommended to use a *static* build of the Poco libraries. If you don't, the changes of someone being able to use the packages executables is really small.

Execute

```
cpack -G DEB
```

#### Building RPM Packages

Execute

```
cpack -G RPM
```

This requires rpmbuild.

### 4.2.2 Building the documentation

The documentation is written in ReStructuredText and built with sphinx, it can be compiled into HTML by

```
cd docs
make
```

## 4.3 Example Configurations

**Note:** The complete list of all configuration options can be found on *Configuration Files*

**Note:** All good ASCII art is courtesy of Joan Stark, all bad ASCII art is me changing her work for the worse.

### 4.3.1 Example 1

The easiest possible setup: a single source and only one store:

```
                   .----.
       .---------. | == |
       |.-"""""-.| |----|
       ||       || | == |
       ||       || |----|
       |'-.....-'| |::::|
       `"")---(""` |___.|
      /:::::::::::\" _  "
     /:::=======:::\`\`\
jgs `"""""""""""""`  '_'

         1 Source
         1 Store
```

The file `sources.config` will contain something like:

```
[ entireHomeDirectory ]
Include[] = /home/ives
Exclude[] = *.bak
Exclude[] = *~
ExecuteAt = every wednesday at 21:00h
```

The file `stores.config` will contain something like:

```
[ externalStore ]
Location = /media/usbDrive/backup
```

This will backup all files in /home/ives, except for those matching `*.bak` or `*~` on every Wednesday at 8 p.m. It will store backups for them in `/media/usbDrive/backup`.

### 4.3.2 Example 2

A setup where there is one “server” and one “workstation”. All backups for the workstation will be stored on the server. The server itself does not have any backup sources.

```
       WORKSTATION                    SERVER
                   .----.            _________
       .---------. | == |           | _______o|
       |.-"""""-.| |----|           ||_______||
       ||       || | == |           | _______ |
       ||       || |----| ========= ||_______||
       |'-.....-'| |::::|           |---------|
       `"")---(""` |___.|           |       O |
      /:::::::::::\" _  "           |       | |
     /:::=======:::\`\`\            |       | |
jgs `"""""""""""""`  '_'            |:::::::::|

         1 Source                    0 Sources
         0 Store                     1 Store
```

#### On the Workstation

The file `sources.config` will contain something like:

```
[ entireHomeDirectory ]
Include[] = /home/ives
Include[] = /home/naomi/importantFile
Exclude[] = /home/ives/movies
ExecuteAt = every 6 hours
```

No `stores.config` is present on the workstation.

#### On the Server

The file `stores.config` will contain something like:

```
[ externalStore ]
Location = /media/bigdrive/
```

#### Conclusion

This setup will backup all files and directories in `/home/ives`, except for those in `/home/ives/movies`. It will also backup the file `/home/naomi/importantFile`. The backup will run every six hours. The workstation and server obviously have to be connected to the same LAN.

CHAPTER

FIVE

# INDICES AND TABLES

- *genindex*
- *search*